Universidade do Porto

Faculdade de Engenharia

FEUP

Aplicação de ID3 ou C4.5 ao diagnóstico de doença renal crónica

Relatório Final

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do grupo:

Luís Oliveira - 201304515 - up201304515@fe.up.pt

Miguel Pereira - 201305998 - up201305998@fe.up.pt

Marta Lopes - 201208067 - ei12106@fe.up.pt

29 de Maio de 2016

Conteúdo

1 Objetivo

Este projeto, desenvolvido na unidade curricular de Inteligência Artificial, consiste na aplicação do algoritmo C5.0 para o diagnóstico da Doença Renal Crónica.

O programa deverá, através do algoritmo C5.0 e com base num conjunto de dados previamente disponibilizados, determinar uma Árvore de Decisão que traduz as regras de classifcação desse conjunto de dados. Estes dados vão ser então analisados de forma a verificar a eventual necessidade de pré-processamento. O modelo obtido deve poder depois ser utilizado na classificação de novos casos.

O objetivo final deste projeto será a determinação da árvore de decisão que traduz essas regras no diagnóstico da doença.

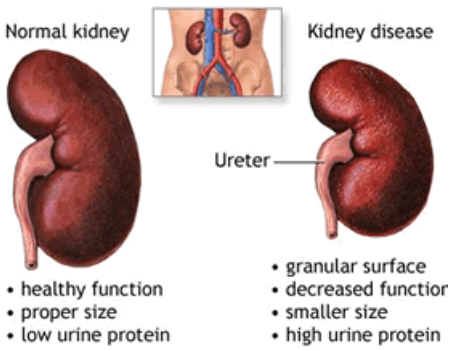
2 Especificação

2.1 Análise ao tema

2.1.1 Detalhes do tema

Este projeto incidiu sobre a Doença Renal Crónica. Esta doença provoca alterações da estrutura ou das funções dos rins, com ou sem alteração da filtração glomerular, por um período maior do que 3 meses e com implicações para a saúde do indívudo.

Para identificar a lesão renal, são utilizados marcadores (ou seja, sintomas) que o paciente vai ter, que vão permitir diagnosticar a doença: a presença de albuminúria maior que 30mg/24h, anormalidades no sedimento urinário, distúrbios eletrolíticos e outras desordens relativas à doença dos túbulos renais, anormalidades detetadas por biópsia renal, exames de imagem, antecedentes de transplante renal e taxa de filtração glomerular menor que 60ml/min.



2.1.2 Ilustração de cenários

Esta ferramenta, criada por nós, poderá ser usada num cenário em que um médico necessitaria de apoio no diagnóstico da doença. O médico poderia usar esta aplicação para confirmar o diagnóstico da doença de um paciente, ou para esclarecer qualquer dúvida no diagnóstico final.

Achamos, também, que seria útil no auto-diagnóstico de qualquer indivíduo que quisesse ter informações sobre o seu estado de saúde, mais especificamente em relação à Doença Renal Crónica.

2.1.3 Training data set

Este conjunto de dados foi obtido no ano passado, após 2 meses de estudo nos Hospitais *Apollo*, na Índia. Existem ao todo 25 atributos no *data set*, dos quais 11 são numéricos e 14 nominais. Neste conjunto de dados vão existir 300 entradas, das quais 200 são indivíduos diagnosticados com a Doença Renal Crónica e 100 são saudáveis. Este conjunto de dados vai ser usado para a ferramenta conseguir determinar uma Árvore de Decisão para diagnosticar a doença. O *training data set* terá esta estrutura:

```
48,80,1.020,1,0,?,normal,notpresent,notpresent,121,36,1.2,?,?,15.4,44,7800,5.2,yes,yes,no,good,no,no,ckd
44,70,1.025,0,0,normal,normal,notpresent,notpresent,?,?,?,?,?,13.8,48,7800,4.4,no,no,no,good,no,no,notckd
```

2.1.4 Test data set

O *test data set* foi obtido de maneira semelhante ao *training data set*. Tem um total de 100 entradas, das quais 50 são indivíduos diagnosticados com a Doença Renal Crónica e 50 são saudáveis. Este conjunto de dados vai ser onde o algoritmo será testado de forma a saber se avalia corretamente se o indíviduo é saudável ou portador da doença, de acordo com as regras de decisão criadas a partir do *training data set*. A estrutura do *test data set* será semelhante à do *training data set* mas com elementos diferentes.

2.1.5 Diagnóstico

Em *cases* vai ser possível ter um ou mais elementos com certos atributos sem a informação relativa a se o elemento é portador da doença ou não. O algoritmo vai então, a partir da árvore de decisão e *rulesets* previamente criados, prever se o elemento tem a Doença Renal Crónica com um certo grau de certeza. A sua estrutura será então a seguinte:

```
41,80,1.020,0,0,normal,normal,notpresent,notpresent,122,25,0.8,138,5.0,17.1,41,9100,5.2,no,no,no,good,no,no,?
```

2.2 Abordagem

2.2.1 Técnicas

Boosting

Esta técnica, incorporada no C5.0 e baseada no trabalho de Rob Schapire e Yoav Freund, vai gerar vários classificadores em vez de apenas um. Quando um caso novo precisa de ser classificado, cada classificador vota na classe onde prevê que esse caso pertence. Os votos são então contados na sua totalidade para determinar a classe final.

Primeiramente, a árvore de decisão é gerada a partir dos casos de treino. O classificador, geralmente, vai classificar erradamente alguns dos casos de treino. É então criado um segundo classificador que vai prestar mais atenção aos casos que foram classificados incorretamente, de forma a tentar que sejam classificados de forma correta pela nova árvore de decisão. O segundo classificador irá ser então diferente do primeiro, contudo, o segundo classificador também poderá cometer classificações erradas, e estas tornam-se foco de atenção durante a construção do terceiro classificador e assim sucessivamente até ser atingido um número pré determinado de iterações ou até o classificador mais recente ser extremamente preciso. Podemos então usar esta técnica usando a flag **-b**.

Winnowing

Nem sempre todos os atributos fornecidos são relevantes para a construção da árvore de decisão e do *ruleset*. Portanto, é útil pré-selecionar o conjunto de atributos que serão utilizados para a construção dos mesmos.

O algoritmo C5.0, através do mecanismo de *winnowing*, consegue prever quais atributos serão ou não relevantes na classificação, fazendo uma estimativa do factor pelo qual o *true error rate* ou o custo de uma classificação errada iria aumentar caso o atributo fosse excluído. No entanto, esta estimativa serve apenas como guia e não deve ser levada literalmente.

Uma vez que este pode ser um processo moroso, é recomendado o seu uso para lidar com *datasets* de grandes dimensões, quando existir razão para suspeitar que muitos dos atributos têm pouca relevância para a tarefa de classificação. Este método pode ser usado no nosso projeto usando a flag **-w**.

Poda da Árvore de Decisão

O algoritmo C5.0 constrói árvores de decisão em duas fases: primeira-

mente, uma árvore grande é construída para classificar todos os dados, e posteriormente é podada para remover partes que possam ter uma taxa de erro relativamente elevada. Este processo de poda é aplicado primeiro em cada sub árvore para decidir se essa vai ser substituída por uma folha ou um sub ramo, e finalmente analisa-se a performance da árvore como um todo. A desativação da poda resulta em árvores de decisão maiores e em algumas aplicações pode ser benéfico. Para isso, neste programa podemos então usar a flag **-g** para o fazer.

2.2.2 Algoritmos e sua breve explicação

O algoritmo C4.5 é um algoritmo de aprendizagem supervisionada, o que significa que aprende sabendo o resultado. Um supervisor diz-nos qual vai ser o resultado final e com base nesses resultados vamos aprender construindo uma árvore, pegando no histórico que permite extrair as regras escondidas nesse histórico. Este algoritmo é uma extensão do ID3 que vai fazer os cálculos de maneira diferente e permite fazer cálculos mesmo quando a amostra tem valores desconhecidos. Vai analisar, ainda, quão bem distribuídos estão os valores da amostra, quer nos atributos, quer na amostra, relativamente aos valores com que se vai classificar a classe. A partir do C4.5 foi então criado o C5.0 que tem algumas melhorias.

O algoritmo funcionará então como se vê na explicação seguinte:

Input: Exemplo, Atributo Alvo, Atributo.

Output: Árvore de Decisão.

Algoritmo:

1. Verificar classe base
2. Construir uma *decision tree* (DT) usando o *training data set*
3. Descobrir o atributo com **maior ganho de informação**
4. Para cada atributo ti que pertença a D, aplicar a DT para determinar a sua classe uma vez que a aplicação de um dado tuplo para uma DT é relativamente simples.

Casos base:

1. Todos os exemplos do *training data set* pertencem à mesma classe, isto é, uma folha da árvore marcada com essa classe é retornada;
2. Estando o *training data set* vazio, é retornada uma folha caracterizada por insuficiência;
3. A lista de atributos, estando vazia, é devolvida uma folha contendo a classe mais frequente ou a disjunção de todas as classes.

2.3 Métricas

A grande dificuldade deste algoritmo baseia-se em descobrir qual o algoritmo pelo qual se vai exercer a decisão. Este atributo pode ser o inicial ou um intermédio, e o método pelo qual se escolhe está presente na métrica que o algoritmo contempla.

A métrica utilizada pelo algoritmo C5.0 é o **ganho de informação** que consiste em obter o maior número de elementos dos conjuntos puros. Estes conjuntos são aqueles em que os resultados se direcionam todos para o mesmo lado, isto é, ou apontam todos para o **sim** ou apontam todos para o **não**. A **pureza** de um cojunto é obtida a partir da medição da **incerteza** de uma classe e esta é dada a partir do cálculo da *entropia* da sua divisão.

A fórmula de ganho será então:

$$Gain(S,\ A)\ =\ H(S)\ -\ \sum_{V\ \in Values(A)} \frac{|Sv|}{|S|} H(Sv)$$

V - Possíveis valores de **A**.
S - Conjunto de exemplos **{Y}**.
Sv - Subconjunto onde $\mathbf{Ya} = \mathbf{V}$.
H(S) - Cálculo da entropia.

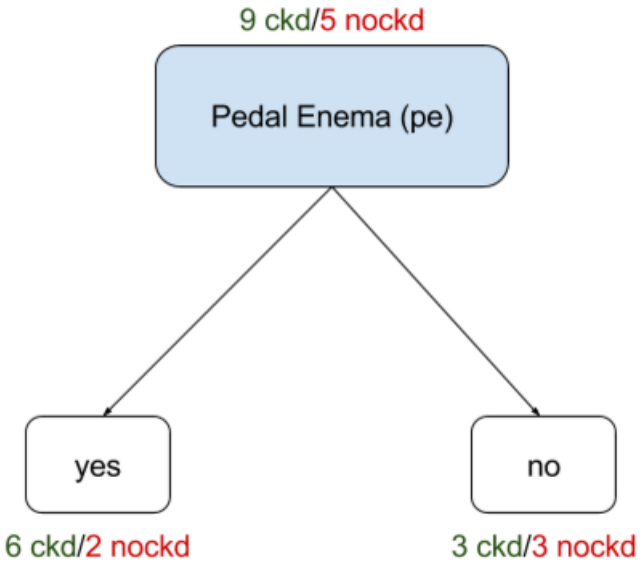
A entropia devolverá o número de *bits* necessários para dizer se Y é positivo ou negativo, o valor da entropia varia entre 0 e 1, sendo que 1 é quando a incerteza é maior. A entropia é então calculada pela seguinte fórmula:

$$H(S)\ =\ -\ p_+ * log_2 p_+\ -\ p_- * log_2 p_-$$

S - Subconjunto de exemplos de treino.
$\mathbf{p_+/p_-}$ - Percentagem de exemplos positivos/negativos em **S**.

2.3.1 Exemplo

Por exemplo, queremos concluir se um paciente tem ou não Doença Renal Crónica (*Chronical Kidney Disease*) e a partir do *training data set* são nos dados 14 pacientes, em que 5 são saudáveis e 5 são portadores da doença.



Queremos então calcular o ganho de informação a partir do atributo **pe** que nos diz se o paciente tem ou não um enema pedal (*pedal enema*).

$$H(S) = -\frac{9}{14} \times log_2 \times \frac{9}{14} - \frac{5}{14} \times log_2 \times \frac{5}{14} = 0.94$$

$$H(S_{yes}) = -\frac{6}{8} \times log_2 \times \frac{6}{8} - \frac{2}{8} \times log_2 \times \frac{2}{8} = 0.81$$

$$H(S_{no}) = -\frac{3}{6} \times log_2 \times \frac{3}{6} - \frac{3}{6} \times log_2 \times \frac{3}{6} = 1.0$$

$$Gain(S, pe) = H(S) - \frac{8}{14} \times H(S_{yes}) - \frac{6}{14} \times H(S_{no}) =$$
$$= 0.94 - \frac{8}{14} \times 0.81 - \frac{6}{14} \times 1.0 = 0.049$$

Este atributo terá então um ganho de informação de 0.049, ou seja, ganharíamos 0.049 *bits* de certeza. Se for dos atributos com maior ganho de informação seria escolhido para a divisão da árvore de decisão.

2.4 Heurísticas

A heurística de seleção de teste original baseada em ganho de informação não era suficiente, era necessário resolver alguns problemas que surgiam quando os atributos se dividiam em vários subconjuntos, tornando-se únicos e ficando assim com um nível de pureza máximo, penalizando os atributos que tivessem um menor número de resultados. Isto pode levar a árvores de decisão que não conseguiriam generalizar e assim poderiam não chegar a nenhuma solução quando confrontada com novos dados. Para contrariar estes problemas foi então criado o **GainRatio** que é realizado posteriormente ao **Gain** e utiliza outra fórmula (**Split**) para prever os problemas anteriormente mencionados, penalizando os atributos que tiverem um maior número de resultados.

$$Split(S,\ A)\ =\ -\ \sum_{V\ \in\ Values(A)} \frac{|Sv|}{|S|} log \frac{|Sv|}{|S|}$$

A - Atributo candidato
V - Possíveis valores de **A**.
S - Conjunto de exemplos **{Y}**.
Sv - Subconjunto onde **Ya = V**.

$$GainRatio(S,\ A)\ =\ \frac{Gain(S,\ A)}{Split(S,\ A)}$$

Esta heurística é utilizada no algoritmo C5.0.

2.4.1 Representação do conhecimento

Os classificadores construídos pelo C5.0 poderão então ser avaliados com os dados de treino, a partir dos quais foram gerados, e também com os dados de teste, não vistos previamente.

Para além dos resultados finais também é feito o *output* da árvore de decisão da seguinte forma:

```
sc > 1.2: ckd (159.9/1.6)
sc <= 1.2:
:...dm = yes: ckd (15.4)
    dm = no:
    :...hemo <= 12.7: ckd (15.8/0.3)
        hemo > 12.7:
        :...pe = yes: ckd (4.4)
            pe = no:
            :...sg = 1.005: notckd (0)
                sg in {1.010,1.015}: ckd (6.1/0.1)
                sg in {1.020,1.025}: notckd (98.5/0.4)
```

A forma de interpretar seria "se sc maior que 1.2 então o paciente terá Doença Renal Crónica". O valor 159.9 representa o número de casos no ficheiro *ck.data* que estão a ser usados naquela folha e 1.6 será o número de casos que estão a ser incorretamente classificados pela folha. Se o paciente não tiver sc maior que 1.2 então passa para outra folha e é escolhido o próximo atributo para ser avaliado e assim sucessivamente até serem avaliados todos os casos.

Os resultados da árvore de decisão dos casos em **ck.data** serão dados da seguinte forma:

Árvore de decisão com 300 casos	
Tamanho	Erros
6	3 (1.0%)

A coluna da esquerda mostra o tamanho de folhas da árvore que não estão vazias. A coluna da direita mostra o número e a percentagem de casos mal classificados. Neste caso a árvore tem 6 folhas e classifica erradamente 3 dos 300 casos, o que implica uma taxa de erro de 1.0%.

A performance dos casos de treino, é ainda analisada numa matriz de confusão que aponta que tipos de erros foram feitos.

(a)	(b)	<- classificado como
200		(a): classe ckd
3	97	(b): classe notckd

Neste caso, sendo que (a) são os portadores de doença e (b) os indivíduos saudáveis, a árvore de decisão classificou erradamente 3 casos saudáveis (classificou como portador da doença).

Para algumas aplicações, especialmente aquelas com muitos atributos, pode ser útil saber quanto é que cada atributo contribui para o classificador. Isso é indicado depois da matriz de confusão:

Attribute usage:
96% sc
49% dm
39% hemo
38% pe
36% sg

A percentagem associada a cada atributo corresponde à percentagem de casos de treino em **ck.data** para os quais o valor desse atributo é conhecido e é usado para prever uma classe. Os atributos que têm *attribute usage* menores que 1% não aparecem nesta lista.

Se **ck.test** existir (ficheiro que contém o *test data set*), a performance do classificador é sumarizada num formato semelhante ao dos casos de treino.

Árvore de decisão com 100 casos	
Tamanho	Erros
6	2 (2.0%)

(a)	(b)	<- classificado como
50		(a): classe ckd
2	48	(b): classe notckd

Finalmente se um ficheiro *.cases* for usado para prever o diagnóstico de um certo indíviduo, irá ser retornada a seguinte tabela:

Nr. Caso	Classe	Previsão
1	?	notckd [0.99]

Com a árvore de decisão previamente criada com os ficheiros *.data e .test* foi então possível prever com 0.99 de certeza que este indíviduo não é portador da Doença Renal Crónica.

3 Desenvolvimento

3.1 Ferramentas e Ambientes de Desenvolvimento

O sistema operativo utilizado para desenvolver o projeto foi Linux Mint 17.1 - Cinnamon. A linguagem de programação utilizada foi C++, usando como IDE o Eclipse Mars 2. Decidimos usar a ferramenta See5 para o *data mining*, que utiliza o algoritmo C5.0 e permite descobrir padrões que formam categorias e constroem classificadores para prever os resultados sobre o domínio em análise.

3.2 Detalhes Revelantes

Decidimos então escolher o algoritmo C5.0, que tem como base o algoritmo C4.5 dado nas aulas, mas que tem algumas evoluções. Os *rulesets* no algoritmo C5.0 vão ter **taxas de erro** menores e o uso de **memória** vai ser bastante mais eficiente na construção dos *rulesets*. Os *rulesets* vão ser também menores no C5.0, tal como as **árvores de decisão** serão mais simples e coesas. A **velocidade** também será bastante maior no algoritmo C5.0 conseguindo acabar em 73 segundos uma tarefa que demoraria 9 horas a terminar no algoritmo C4.5.

Neste algoritmo vai ser ainda possível a utlização de **boosting** para construir várias árvores de decisão, tentando melhorar as anteriores, fazendo assim previsões mais precisas. No C5.0 é possível também ponderar diferentes atributos e tipos de classificações incorretas. A ferramenta permite ainda resolver a questão do *label noise*, que ocorre quando os valores dos atributos de duas instâncias são exatamente iguais, mas diferem do resultado final.

No nosso projeto foram utilizadas ainda várias *flags* que vão alterar a forma como vão ser obtidos os resultados, podendo ser usadas em simultâneo:
-b : utiliza boosting
-w : utiliza winnowing
-g : não utiliza poda

Estes três métodos já foram explicados previamente na secção 2.2.1.

4 Experiências

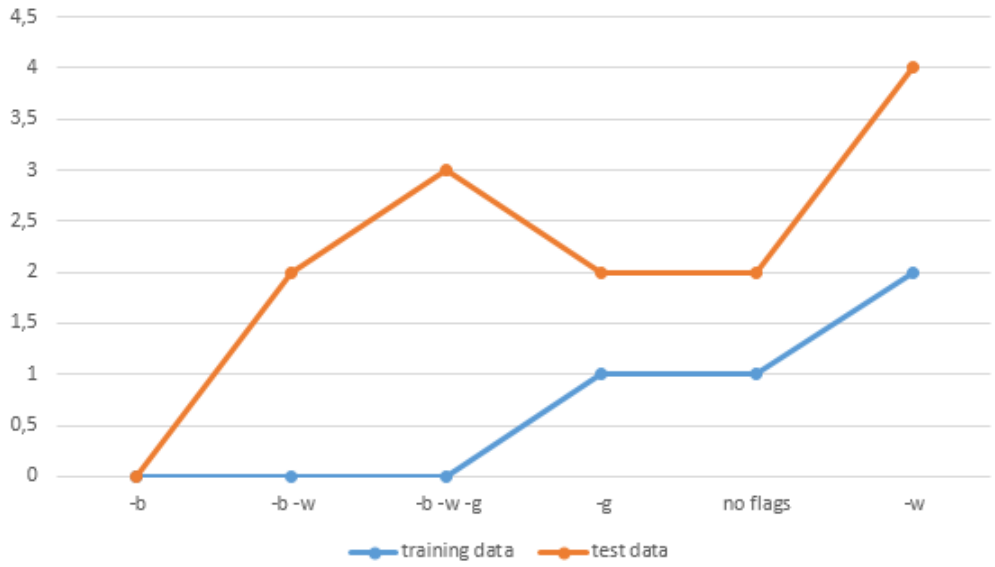
4.1 Objetivos

O objetivo destas experiência será obter a árvore de decisão com base nos *data sets* fornecidos, e saber quais dos métodos avaliados será mais preciso, ou seja, terá menor taxa de erros aquando da classificação dos elementos.

4.2 Resultados

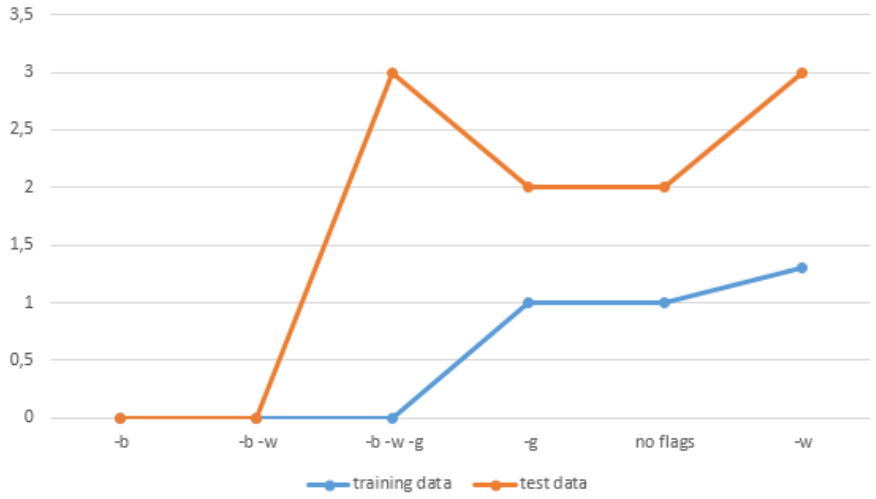
Primeiramente obtivemos o nosso *output* **considerando todos os atributos** que foram inicialmente inseridos no *data set*. Em baixo podemos ver os resultados obtidos, de acordo com a taxa de erros, organizados, numa tabela e num gráfico, do melhor para o pior:

	-b	**-b -w**	**-b -w -g**	**-g**	**no flags**	**-w**
train data	0%	0%	0%	1%	1%	2%
test data	0%	2%	3%	2%	2%	4%



Após um estudo aprofundado sobre a Doença Renal Crónica e dos seus sintomas e também do impacto dos atributos na construção da árvore de decisão e na diminuição nas taxas de erro, **foram ignorados alguns atributos**, sendo assim possível **diminuir a taxa de erros** de forma geral. Estes resultados podem ser visualizados em baixo numa tabela e num gráfico, também ordenados do melhor para o pior.

	-b	-b -w	-b -w -g	-g	no flags	-w
train data	0%	0%	0%	1%	1%	1.3%
test data	0%	0%	3%	2%	2%	3%



5 Conclusão

Podemos então concluir que a taxa de erro no geral é bastante baixa, comparando com outros tipos de *data sets*. Isto deve-se ao facto dos atributos para o diagnóstico da Doença Renal Crónica serem amostras de sangue, o que torna a previsão bastante precisa, visto que os valores obtidos vão contribuir para um fácil estudo da doença. Ainda assim, a flag de **boosting** permite ao algoritmo ter uma taxa de erro nula, o que demonstra que esta técnica é bastante útil. A técnica de **winnowing**, por outro lado, aumenta a taxa de erro, o que nos diz que este algoritmo não está a verificar de forma correcta, para este *data set*, quais os atributos relevantes para a construção das árvores de decisão. A **desativação da poda da árvore** também não melhora a taxa de erro, ou seja, neste caso é necessário que a árvore de decisão use a técnica de poda avançada.

Depois de terminado o projeto, o grupo conclui que esta abordagem ao algoritmo C5.0 mostra que cada vez há mais avanços no que diz respeito a algoritmos de tratamento de dados. A nível de aprendizagem podemos dizer que, apesar de usarmos o C5.0, foi possível entender a lógica de funcionamento dos algoritmos precendentes: C4.5 e ID3. Podemos também dizer que é relativamente fácil e rápido obter este tipo de resultados com a *framework* que utilizamos e que por isso tivemos que fazer um estudo adicional para per-

ceber melhor este algoritmo. Concluimos também que é dificil saber quais os atributos que são desnecessários, sendo necessários fazer alguns testes para perceber quais aumentam ou diminui a taxa de erro, e quais são, de acordo com a ferramente, mais usados para a árvore de decisão.

Para além disso, achamos que este tipo de aplicações podem ter bastante potencial para serem usadas na previsão de vários casos, nomeadamente na medicina.

6 Melhoramentos

Apesar de estarmos satisfeitos com o que fizemos para este projeto, achamos que uma das possíves melhorias a fazer seria criar uma *interface* que facilitaria o uso da ferramenta e permitiria a sua distribuição para o público no geral. Outro melhoramento que poderia ser implementado seria facilitar a visualização dos *outputs* e da árvore de decisão simplificando o estudo da nossa aplicação e dos seus resultados.

7 Recursos

7.1 Bibliografia

See5/C5.0: *https://www.rulequest.com/r210.html*
C5.0: An Informal Tutorial: *https://www.rulequest.com/see5-unix.html*
C5.0 Algorithm to Improved Decision Tree with Feature Selection and Reduced Error Pruning

7.2 Software

Linux Mint 17.1 - Cinnamon
Eclipse Mars 2
TeXworks: *https://www.tug.org/texworks/*
Github: *https://www.github.com*
Table Generator: *http://www.tablesgenerator.com/*

7.3 Contribuição de cada elemento

Luís Oliveira: 33.3%
Miguel Pereira: 33.3%
Marta Lopes: 33.3%

8 Apêndice

Deverá ser feito o download da *framework* para *Linux* em *https://www.rulequest.com/GPL/C50.tgz* para uma pasta denominada c50 que deverá encontrar-se dentro da pasta do projeto, de seguida será necessário abrir o terminal e ir para a pasta extraída e correr o comando *make* para compilar. Na pasta prediction irá existir a ferramenta adicional (obtida em *http://www.rulequest.com/see5-public.tgz*) para previsão de casos que deverá ser compilada com o comando *gcc sample.c -lm*. Finalmente, com a *framework* e a ferramenta adicional corretamente instaladas será possível gerar os classificadores correndo o programa, escolhendo se se quer usar o *training* e *test data set*, ou se apenas se quer usar o *training data set*. Poderão ser escolhidas várias opções que vão afetar o tipo de classificador que o C5.0 vai produzir, bem como a forma como o classificador é construído.
Após gerada a árvore de decisão, é também possível diagnosticar um paciente, escolhendo a opção *Make a diagnosis*. O paciente ou pacientes que se pretenderem diagosticar deverão ser colocados num ficheiro *.cases* com os atributos pretendidos.